Securing Embedded Software using Software Dynamic Translation

Position Paper for ARO Planning Workshop on
Embedded Systems and Network Security
February 22–23, 2007

Jack W. Davidson and Jason D. Hiser, University of Virginia, {jwd, jdh8d}@virginia.edu

1. Introduction

Embedded computer systems have become key building blocks of our nation's vital infrastructure. Critical systems controlled by embedded computer systems include communications systems, transportation and navigation systems, financial systems, medical systems, power distribution systems, and critical defense systems. Failure or compromise of such systems can have significant consequences including disruption of critical services, financial loss, and loss of life. Because critically functionality in embedded systems is increasingly implemented via software, three important research challenges for securing these systems is to provide protection from malicious observation, making them tamper resistant, and making them more resilient to unintentional and intentional memory errors in unsafe code that could be used to compromise an embedded system.

Unfortunately, securing embedded systems present several unique challenges not found in typical desktop or enterprise systems. Because of cost and power considerations, the execution environment for embedded software is often resource constrained—CPUs have limited processing power, there is often no memory management unit, and memory space is limited. Furthermore, embedded systems are frequently deployed in the field and must operate in physically insecure environments.

In this position paper, we discuss software dynamic translation and its potential for protecting software from malicious observation and tampering. While software dynamic translation can also be used to provide protection from unintentional and intentional memory errors that can be used to compromise an embedded system, even a brief discussion of the needed research and challenges in that area is beyond the scope of this paper.

2. Malicious Observation and Tampering

A trend in embedded systems is to provide functionality, which in the past was usually provided by hardware, via software. There are many advantages to using software instead of hardware to provide required functionality—reduced cost, increased flexibility, the ability to provide enhancements and patches, etc. However, moving functionality from hardware to software provides malicious parties easier access to valuable *intellectual property* (IP). In the context of this position paper, IP means information that an adversary could use for some malicious purpose (e.g., maliciously modifying a system, discovering a weakness that could be used to disable the system or render it ineffective, etc.) *Malicious observation* is the process of obtaining valuable IP. Of course, malicious observation could also be used to obtain valuable IP for commercial or financial advantage. Closely related to malicious observation is *malicious tampering*. Malicious tampering is the modification of software to change its intended behavior to achieve some malicious goal (e.g., cause damage, render the system ineffective, subvert some safeguards or licensing checks, etc.). Obviously, to intelligently tamper with a system, an attacker must have some knowledge about the operation of the system. Consequently malicious observation and tampering are closely related.

Because embedded systems are often deployed in hostile or insecure environments, one must assume that an attacker can gain physical access to the system. Consequently, an attacker can employ a variety of means to maliciously observe the operation of the software including the use of a virtual execution environment. The adversary can inspect, modify, or forge any information in the system. An adversary can run the program repeatedly and aggregate information from multiple runs of the program. In this extremely harsh environment, the adversary "holds all the cards" and with adequate time and resources, can gain a detailed understanding of the operation of the system.

3. Fundamental Limitations of Current Approaches

Current approaches to thwarting malicious observation have focused on making software hard to analyze statically. Addressing dynamic approaches to malicious observation has received little attention. While hardware approaches to preventing malicious observation and tampering can be effective in some contexts, hardware approaches may not be feasible within the cost- and resource-constraints imposed on embedded systems. In a similar vein, the few software approaches that have been proposed can require considerable computational resources and therefore are not applicable to embedded systems. Finally, much previous work has assumed an unrealistic threat model where an attacker does not have unfettered access to the system.

4. Software Dynamic Translation

A promising approach for addressing the very difficult problem of securing embedded software from malicious observation and tampering is to use software dynamic translation (SDT). SDT is a technology that enables software malleability and adaptivity at the binary instruction level by providing facilities for monitoring and dynamically modifying a program as it executes. SDT can affect an executing program by injecting new machine code, modifying existing code, or by monitoring and changing the control flow of the executing program. SDT has been successfully used in a variety of areas including binary translation, fast machine simulation, dynamic optimization, and to protect software from attacks that inject malicious code or attempt to change the normal execution flow of the program.

Using SDT, we envision a three-pronged approach to address this difficult challenge. First, SDT coupled with strong encryption technology can be used to make it difficult and costly for an adversary to statically and dynamically analyze embedded software. However, with physical access to the system and with adequate resources, a sophisticated and determined attacker could eventually obtain a detailed understanding of the software's operation. Therefore our second approach is to use SDT to make it difficult for an attacker to examine or modify a running system (including the one that an attacker might have obtained for malicious observation). Third, SDT is used to create diverse versions of the software to make it difficult to aggregate information across different executions of the system. Dynamic diversity also ensures that knowledge gained by capturing and observing one instance of a system is not applicable to any other deployed instance. Thus, even if an attacker can determine what modifications to make to achieve their goal for one instance of the system (i.e., the system to which they have physical access), this knowledge is not useful for attacking other instances of that system.

5. Milestones

For SDT to be applicable to embedded systems, it must be demonstrated that SDT can be applied to embedded software running on typical embedded processors. Preliminary results for the ARM processor using some widely used embedded benchmarks indicate that SDT can be efficiently accomplished on an embedded system. Research adapting SDT to other architectures and other types of systems (e.g., hard real-time systems, reactive systems, etc.) needs to be carried out. Of paramount importance is the ability to perform SDT on resource-constrained systems without excessive overhead.

A difficult problem is the assessment of the effectiveness of techniques to provide protection against malicious observation and tampering. Of particular concern is that there are no objective metrics or models for assessing the effectiveness of techniques to protect software against malicious observation when an intelligent human adversary is guiding the effort (which will almost always be the case). Development of metrics and models for assessing the effectiveness of techniques used to protect embedded software against malicious observation and tampering is critical.

While hardware solutions to the malicious observation and tampering problem have been proposed, they can greatly increase the cost of a system. For systems where deployment means the delivery of thousands of devices, the cost of a comprehensive hardware solution may not be feasible. However, modest hardware additions designed to support SDT-based solutions to malicious observation and tampering may be cost effective and provide a higher level of protection than simple hardware- or SDT-based protection mechanisms alone.

Securing Embedded Software using Software Dynamic Translation

Jack W. Davidson and Jason Hiser

University of Virginia

Problem

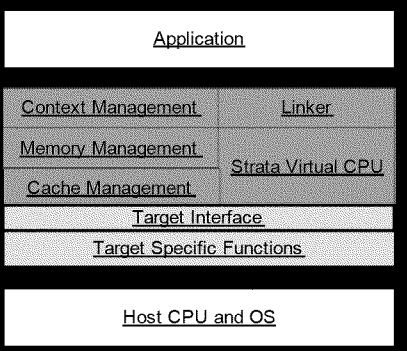
- Embedded systems key building blocks of nation's vital infrastructure
  - Communication systems
  - Transportation and navigation systems
  - Financial systems
  - Power distribution systems
  - Defense systems
  - Etc.
- System functionality is increasingly provided by software instead of hardware
- Must protect the software in these systems from malicious observation and tampering

Threat model

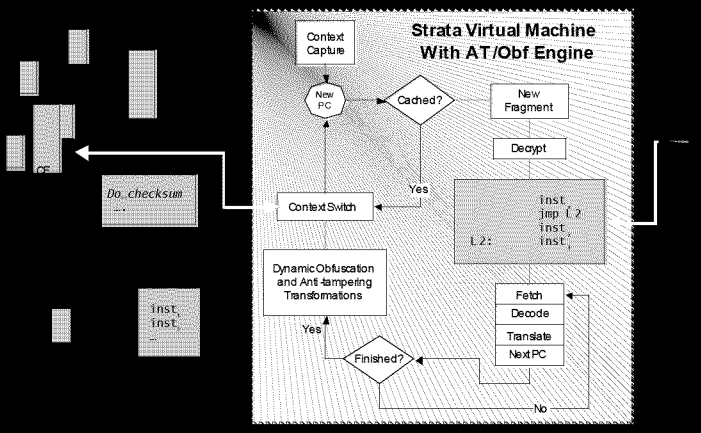
- Adversary has physical access to system
- Adversary controls execution environment
  - Execute directly and observe
  - Simulate and observe
  - Provide false inputs
  - Run repeatedly
  - Use sophisticated dynamic analysis tools
- White-box attack where the adversary "holds all the cards"
  - Example, HD protection recently cracked (http://www.theregister.co.uk/2007/02/14/aacs_hack/)

Our Approach: Software Dynamic Translation

- Any software that *intercepts, controls, or modifies* a program as it runs
- Subsumes:
  - –Dynamic optimization / compilation
  - –Dynamic binary translation
  - –Dynamic instrumentation (e.g., profiling)
  - –Host virtualization
  - –Debugging



Using SDT for Obfuscation and Anti-tampering



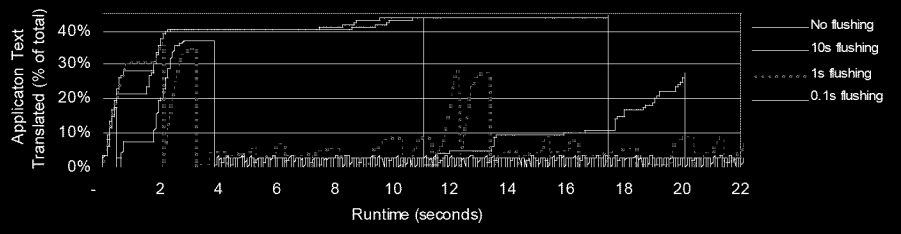
Benefits

- Prevents static disassembly and analysis
  - Code is encrypted on disk
  - Must run SDT system to materialize code
- Provides dynamic obfuscation of code
  - Natural obfuscation of code by SDT system
  - Dynamically apply obfuscations
- Prevents manipulation of running code
  - Guards prevent changing application or SDT system
  - Fragment cache is protected
- Limits leakage of information
  - Flush fragment cache frequently
  - Multiple runs provide less advantage to attacker
- Provides diverse implementations
  - Dynamic transformations applied randomly
  - Weakness or vulnerability discovered in one instance not necessarily exploitable in other instances

Research challenges for anti-tampering in embedded systems

- Develop metrics for evaluating degree of obfuscation and resistance to tampering
- Managing overhead (both space and time) in constrained-resource systems
- Satisfying real-time requirements
- Investigate melding low-cost hardware approaches (suitable for widely deployed embedded systems) and SDT approach
- Many others …

Limiting leakage of information



Runtime Overhead